

|  |
| --- |
|  |

VoQ Switch Abstraction Interface

Change Proposal

|  |  |
| --- | --- |
| **Title** | **VoQ Switch SAI Proposal** |
| **Authors** | **Broadcom** |
| **Status** | **In Review** |
| **Type** | **Standards Track** |
| **Created** | **07/27/2016** |
| **SAI-Version** | **V0.9.3** |

**Contents**

[List of Changes](#_gjdgxs) **1**

[Overview](#_30j0zll) **1**

[**Integrated VoQ Switch**](#_q6avho59xwqz) **1**

[1.1 Packet Flow in VoQ Switch](#_1fob9te) 1

[Chassis Switch vs. Single Device Switch difference](#_2hrqgd7yf179) 2

[1.2 Multi cores devices issues](#_jry5dk6lzett) 3

[1.3 Switch Fabric Modeling](#_j2wvofoujupx) 3

[1.4 SAI Extension proposal](#_mpthmux5wci8) 3

[**SAI Specification**](#_cv0joqjqaojl) **4**

[Switch Configuration Info](#_4o5r3t21j8y8) 4

[SAI\_SWITCH\_ATTR\_TYPE](#_76hu6lid8k6o) 4

[SAI\_SWITCH\_ATTR\_SWITCH\_ID](#_lk4wuubmibu3) 4

[SAI\_SWITCH\_ATTR\_MAX\_SYSTEM\_CORES](#_plkp09nkw2f) 5

[SAI\_SWITCH\_ATTR\_SYSTEM\_PORT\_CONFIG\_LIST](#_a3wd64ue8l6t) 5

[SAI\_SWITCH\_ATTR\_FABRIC\_PORT\_CONFIG\_LIST](#_mf91jiyxatgs) 6

[SAI\_SWITCH\_ATTR\_SYSTEM\_PORT\_LIST](#_ud70e2uc3v5) 7

[SAI\_SWITCH\_ATTR\_FABRIC\_PORT\_LIST](#_oiqb2tda4y2w) 7

[Fabric Port Reachability](#_73gy2qr5ddi6) 8

[Switch Statistics](#_ydjorcqez6py) 8

[Port Attributes, Types, and APIs](#_jnvhvax6xx2r) 9

[New Port Types](#_o5hf0vtl0gik) 9

[Port Connectivity Attributes for Logical, System and Fabric ports](#_xx5k9pve8ody) 10

[Fabric Port Error Status Attributes](#_rhz1npv3gwkp) 11

[SAI\_OBJECT\_TYPE\_LAG](#_ivqc0eem37fz) 11

[New Ports Stats](#_d5fgefuc0b9i) 11

[Queue Attributes, Types, and APIs](#_c1n2f1odf6w) 12

[New Queue Types](#_jvtwvi4waf2y) 12

[VoQ Attributes](#_71rrp66g2nx7) 13

[Fabric Queue Attributes](#_x6vxvfcowdia) 13

[Modification to queue stats](#_32znydl2ezu5) 13

[2.5 New Neighbor Entry Attribute](#_b5t9la1x4g9s) 15

[**SAI Usage Examples**](#_og4c80zdr0bj) **16**

[SAI Switch Create API](#_lj1pg43o4ut2) 16

[SAI Switch Get Attribute API](#_8qdklf26rw2k) 17

[SAI Port Create API](#_e94yujejxtky) 18

[SAI Port Bulk Object Create API](#_6lqwxvc5nvc0) 19

[**Appendix**](#_nbcmbkmuhzgs) **20**

[SAI\_SWITCH\_ATTR\_PORT\_CONFIG\_LIST](#_f82phd3gkbkt) 20

# List of Changes

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Changes | Name | Date |
| 0.5 | Initial draft | Gabi Bracha | 1/16/2020 |
| 0.6 | Made changes to SAI spec sections | Darryl Satterwhite,  Michael Li | 2/06/2020 |
| 0.7 | Update struct changes after SAI metadata checks | Michael Li | 3/11/2020 |
| 0.8 | Replaced SYSTEM\_INDEX with SWITCH\_ID | Israel Meilik, Gabi Brach | 3/12/2020 |

License

© 2014 Microsoft Corporation, Dell Inc., Facebook, Inc, Broadcom Corporation, Intel Corporation, Mellanox Technologies Ltd.

As of September 9, 2014, the following persons or entities have made this Specification available under the Open Web Foundation Final Specification Agreement (OWFa 1.0), which is available at <http://www.openwebfoundation.org/legal/the-owf-1-0-agreements/owfa-1-0>

Microsoft Corporation, Dell Inc., Facebook, Inc, Intel Corporation, Mellanox Technologies Ltd.

You can review the signed copies of the Open Web Foundation Agreement Version 1.0 for this Specification at <http://opencompute.org/licensing/>, which may also include additional parties to those listed above.

Your use of this Specification may be subject to other third party rights. THIS SPECIFICATION IS PROVIDED "AS IS." The contributors expressly disclaim any warranties (express, implied, or otherwise), including implied warranties of merchantability, noninfringement, fitness for a particular purpose, or title, related to the Specification. The entire risk as to implementing or otherwise using the Specification is assumed by the Specification implementer and user. IN NO EVENT WILL ANY PARTY BE LIABLE TO ANY OTHER PARTY FOR LOST PROFITS OR ANY FORM OF INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OF ANY CHARACTER FROM ANY CAUSES OF ACTION OF ANY KIND WITH RESPECT TO THIS SPECIFICATION OR ITS GOVERNING AGREEMENT, WHETHER BASED ON BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), OR OTHERWISE, AND WHETHER OR NOT THE OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE FOLLOWING IS A LIST OF MERELY REFERENCED TECHNOLOGY: Microprocessor technology, semiconductor manufacturing technology, operating system technology (including without limitation networking operating system technology), emulation technology, graphics technology, video technology, integrated circuit packaging technology and the like, compiler technologies, object oriented technology, optical/RF communications technology including chip I/O and driver technology, bus technology, memory chip technology (including, without limitation, NAND memory, NOR memory, resistive RAM (RRAM), seek scan probe (SSP) memory, nonvolatile memory (including without limitation, memory based on chalcogenide materials, phase change memory (PCM), one or more stacked layers of memory cells, embedded PCM memories, non-volatile cache memory, solid state drives, SRAM, embedded DRAM, ferro-electric memory, and polymer memory)) and/or health-related and medical technology. IMPLEMENTATION OF THESE TECHNOLOGIES MAY BE SUBJECT TO THEIR OWN LEGAL TERMS.

# Overview

Definition and Acronyms

|  |  |
| --- | --- |
| VoQ | Virtual Output Queue |
| QoS | Quality of Service |
| WRED | Weighted Random Early Drop |
| SP | Strict Priority |
| WFQ | Weighted Fair Queue |

# Integrated VoQ Switch

A chassis switch is a system consisting of several Line cards populated with Switch devices and Fabric Cards populated with Fabric devices. The Line Cards switches are connected to network ports on one side and to the Fabric devices on the other side. The Fabric device provides full connectivity between all switch devices. A Clos network is a typical example.

Lets refer to the totality of all network ports in all the switches on the line card as the chassis' System-Ports. The chassis operates as a single integrated switch between all system ports. Thus, they provide switches with very high radix ( 100-1000s) that are typically used in higher levels of Data center networks.

There is also a simplified chassis where 2-3 switch devices are connected in a mesh directly with the fabric ports without an intermediate fabric switch.

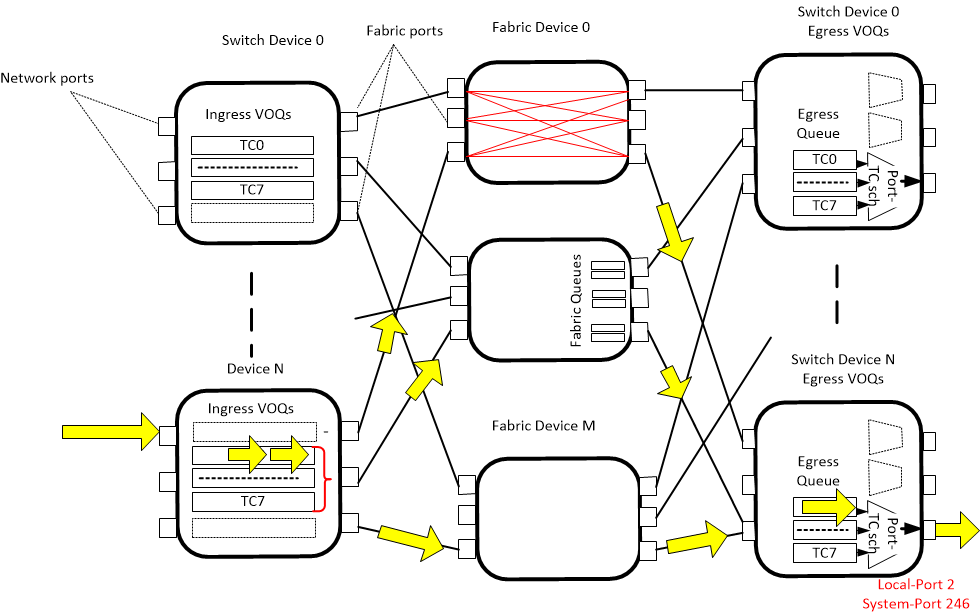
The concepts and APIs presented in this document also apply to a switch consisting of a single device.

Typically, the fabric device is a different and simpler type of device than the line card switch. However, in some implementations a "regular" switching device may be used.

## 1.1 Packet Flow in VoQ Switch

Packet arrives at the network port of a switch device. At the device ingress, the packet is forwarded according to their header, and is allocated a Traffic Class (TC) and a System Port in some switch device (which may be on a remote device or the same device). The packet is placed in a Virtual Output Queue (VoQ) at the ingress that holds only packets directed to that specific VoQ and the packet's TC. Thus, if a system have 20 Line Cards, with 2 devices in each Line Card, each device supports 72 100G ports and a CPU ports, and 8 TCs are supported, then the each switch device ingress must include at least (72+1)\*2\*20\*8 = 23360 VOQs, organized in bundle of 8 VOQs per System-Ports. The packet waits in the VoQs until it is granted fabric access. The agent that grants the fabric access is typically a scheduler at the egress device, that matches the rate of data it summons to the egress to the fabric bandwidth and System-Ports rate. The granting is typically done by sending through the fabric a special control cell (data units) called Credit that allows sending a certain quantity of data, e.g., 2KB. The scheduler has a QOS policy that decides how to allocate bandwidth between the competing TCs. If the port is congested, VoQs with lower priority will not receive fabric transmission grants, their level will rise until they start dropping packet according to a resources management policies (e.g. Tail-Drop, or WRED), or they will be marked according to a Congestion Avoidance protocol (e.g. ECN).

Packets are dequeued from the VOQ and are load balanced across all fabric ports to adjacent Fabric devices. Typically that process involves fragmenting the packets into cells, to ensure efficient load balancing. However, in some chassis implementation entire packets may be sent. The cells take multiple routes through several fabric devices. Typically the fabric devices do not require standard protocols, as they only determine across which ports to send the cells, and they employ proprietary protocol to learn the connectivity between Fabric-Ports and switches devices, and as such require no additional configuration. The cells converge in the switch device egress where they are re-assembled and their order is re-established, then they are placed at (shallow) queue per each network port and TC. Typically each TC has one queue per Unicast traffic and one per Multicast traffic. Each port has a scheduler that arbitrates between competing Egress Queue by QoS policy, typically assigning strict priority or Weighted Fair Queue (WFQ) between the TCs. That policy is essentially the same policy that allocates transmission grants to the VoQs.

****

**Figure 1 VoQ Switch structure and packet walkthrough**

### Chassis Switch vs. Single Device Switch difference

## 1.2 Multi cores devices issues

High bandwidth devices are usually multi-core devices. Each core is associated with a subset of Network ports. Each core has its own data path and control path, as well as a scheduler for its own network ports. The cores share the fabric ports. For example, in a two core device, each VoQ is implemented as two distinct VoQs, one in each core. Thus, when we are performing statistics on a VoQ, we want to hide that internal VoQ partitioning and show the cumulative numbers.

Each port also has two local IDs according to the reference scheme. A port has a switch level ID, and a core level ID. For example a two-cores switch has 64 ports. (Switch) Port 35 is also Port 3 on Core 1

Also System-Port location needs to be more specific, rather than identifying it as {Device-ID, Device-Port-Index} , we identify it as {Device-ID, Core-Index, Core-Port-Index }.

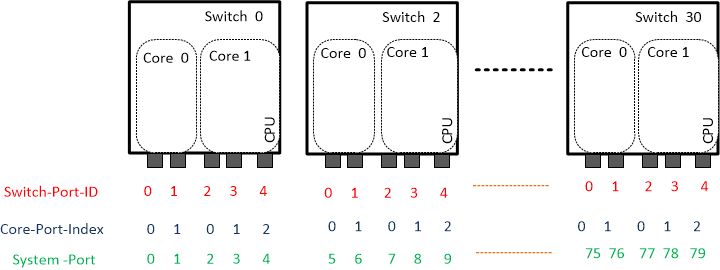
## 1.3 Switch Fabric Modeling

The fabric switch consists of a list of fabric ports. Each port is associated with one to three fabric queues where cells may accumulate. There are counters per device and ports, and per queues. The fabric queue level is encoded to 4 values that indicate congestion level at the fabric queue.

## 1.4 SAI Extension proposal

* Introduce SAI switch attributes to pass chassis specific information to provide a system identifier, the number of total cores in the chassis, and fabric port configuration information.
* Introduce a new Port object subtype “Fabric-Port” for fabric facing ports of the Switch and the ports of the Fabric device.
* Introduce a new SAI object “System-Ports”, representing network or CPU ports in the switch . Note, each local port of the Switch device also has a System-Port instance (though with different IDs).
* A switch has 3 separate lists of ports: Local Network ports, System-Ports, and Fabric-Ports. Note all switches have the same System-Ports list consisting of all Network ports in the entire chassis, including its own ports, and all CPU ports.
* In a VoQ Switch, the SAI\_QUEUE\_TYPE\_UNICAST object refers to the Egress Queues. While the ingress VoQs use the sub-type SAI\_QUEUE\_TYPE\_UNICAST\_VOQ.
* A System-Port comprises the identification of its location in the chassis – Device-ID, Core-Index, and Local-Core-Port-ID, and a list of VoQs, corresponding to the Traffic Classes.
* Each VoQ is associated with a WRED-Profile. Typically Drop and Color statistics are applied to the VoQs (rather than the Egress Queues)
* Fabric port is associated with one to three TX- Fabric-Queues, whose IDs are the object handles for statistics collection.

Following table show a VOQ based system comprising of 16 switches, 2 cores, 2 network ports per each core and a CPU ports and the 3 ID reference systems: Port index in the switch, ports index in the core and the global System Port ID



# SAI Specification

The changes are highlighted in red.

## 

## Switch Configuration Info

## SAI\_SWITCH\_ATTR\_TYPE

A new switch type enum is introduced for VOQ based switch devices and fabric switch devices (ie Ramon).

|  |
| --- |
| /\*\*  \* @brief Attribute data for #SAI\_SWITCH\_ATTR\_TYPE  \*/  typedef enum \_sai\_switch\_type\_t  {  /\*\* Switch type is Switching Network processing unit \*/  SAI\_SWITCH\_TYPE\_NPU,  /\*\* Switch type is PHY \*/  SAI\_SWITCH\_TYPE\_PHY,  /\*\* Switch type is VOQ based NPU \*/  SAI\_SWITCH\_TYPE\_VOQ,  /\*\* Switch type is Fabric switch device \*/  SAI\_SWITCH\_TYPE\_FABRIC,  } sai\_switch\_type\_t; |

## SAI\_SWITCH\_ATTR\_SWITCH\_ID

This switch id represents the switch device’s first core. The external supervisor needs to be aware of the number of cores and skip indexes appropriately. The external supervisor has to have detailed knowledge of cores.

|  |
| --- |
| /\*\*  \* @brief Vendor specific switch id. Identifies switch chip  \*  \* Mandatory in VoQ Switch  \*  \* @type sai\_uint32\_t  \* @flags CREATE\_ONLY  \* @default empty  \*/  SAI\_SWITCH\_ATTR\_SWITCH\_ID |

## SAI\_SWITCH\_ATTR\_MAX\_SYSTEM\_CORES

This defines the max number of cores in the VoQ Switch (a.k.a. Chassis). This can be used for calculation of internal resources.

|  |
| --- |
| /\*\*  \* @brief Maximum number of cores in the VoQ Switch (chassis)  \*  \* @type sai\_uint32\_t  \* @flags CREATE\_ONLY  \* @default empty  \*/  SAI\_SWITCH\_ATTR\_MAX\_SYSTEM\_CORES |

## SAI\_SWITCH\_ATTR\_SYSTEM\_PORT\_CONFIG\_LIST

This switch attribute structure defines a list of system port parameters passed to create\_switch API. There is a system port for every port in the chassis. Ports include front panel ports, cpu port(s), and other internal ports (excluding Fabric Ports). Defined in “saitypes.h”

|  |
| --- |
| typedef struct \_sai\_system\_port\_config\_t {  uint32\_t port\_id;  uint32\_t attached\_switch\_id;  uint32\_t attached\_core\_index;  uint32\_t attached\_core\_port\_index;  uint32\_t speed;  uint32\_t num\_voq  uint32\_t base\_voq\_index;  } sai\_system\_port\_config\_t;  typedef struct \_sai\_system\_port\_config\_list\_t  {  /\*\* Number of entries in the list \*/  uint32\_t count;  /\*\* System port list \*/  sai\_system\_port\_config\_t \*list;  } sai\_system\_port\_config\_list\_t;  /\*\*  \* @brief System port config list.  \*  \* @type sai\_system\_port\_config\_list\_t  \* @flags CREATE\_ONLY  \* @default empty  \*/  SAI\_SWITCH\_ATTR\_SYSTEM\_PORT\_CONFIG\_LIST |

## SAI\_SWITCH\_ATTR\_FABRIC\_PORT\_CONFIG\_LIST

This switch attribute structure defines a list of fabric port parameters passed to create\_switch API. Used to pass fabric port info to SAI. Defined in “saitypes.h”

\*\*\* To-Do: remove all lanes related associations to a separate API proposal \*\*\*

|  |
| --- |
| #define SAI\_MAX\_PORT\_LANES 8  typedef struct \_sai\_fabric\_port\_config\_t {  uint32\_t port\_id; // logical port id  uint32\_t speed; // port speed  uint32\_t count; // actual lanes used  uint32\_t lanes[SAI\_MAX\_PORT\_LANES]; // fixed max number of lanes that can be utilized  } sai\_fabric\_port\_config\_t;  typedef struct \_sai\_fabric\_port\_config\_list\_t  {  /\*\* Number of entries in the list \*/  uint32\_t count;  /\*\* Fabric Port list \*/  sai\_fabric\_port\_config\_t \*list;  } sai\_fabric\_port\_config\_list\_t;  /\*\*  \* @brief Fabric Port config list.  \*  \* @type sai\_fabric\_port\_config\_list\_t  \* @flags CREATE\_ONLY  \* @default empty  \*/  SAI\_SWITCH\_ATTR\_FABRIC\_PORT\_CONFIG\_LIST |

## SAI\_SWITCH\_ATTR\_SYSTEM\_PORT\_LIST

Used in sai\_get\_switch\_attribute API to get the number of system ports and an array of system port objects.

|  |
| --- |
| /\*\*  \* @brief Number of system ports  \*  \* @type sai\_uint32\_t  \* @flags READ\_ONLY  \*/  SAI\_SWITCH\_ATTR\_NUMBER\_OF\_SYSTEM\_PORTS  /\*\*  \* @brief Get the system port list  \*  \* @type sai\_object\_list\_t  \* @flags READ\_ONLY  \* @objects SAI\_OBJECT\_TYPE\_PORT  \* @default internal  \*/  SAI\_SWITCH\_ATTR\_SYSTEM\_PORT\_LIST |

## SAI\_SWITCH\_ATTR\_FABRIC\_PORT\_LIST

Used in sai\_get\_switch\_attribute API to get the number of fabric ports and an array of fabric port objects.

|  |
| --- |
| /\*\*  \* @brief Number of fabric ports on the switch  \*  \* @type sai\_uint32\_t  \* @flags READ\_ONLY  \*/  SAI\_SWITCH\_ATTR\_NUMBER\_OF\_FABRIC\_PORTS  /\*\*  \* @brief Get the fabric port list  \*  \* @type sai\_object\_list\_t  \* @flags READ\_ONLY  \* @objects SAI\_OBJECT\_TYPE\_PORT  \* @default internal  \*/  SAI\_SWITCH\_ATTR\_FABRIC\_PORT\_LIST |

## 

## Fabric Port Reachability

These attributes provide a list of fabric ports on a switch/fabric device that can reach a remote switch device (switch\_id).

|  |  |
| --- | --- |
| typedef struct \_sai\_switch\_fabric\_port\_reachability\_list\_t  {  /\*\* Remote device (SAI\_SWITCH\_TYPE\_NPU) \*/  uint32\_t switch\_id;  /\*\* list of fabric ports that can reach remote switch\_id \*/  uint32\_t count;  sai\_object\_id\_t \*list;  } sai\_switch\_fabric\_port\_reachability\_list\_t;  /\*\*  \* @brief Fabric port reachability list  \*  \* @type sai\_switch\_fabric\_port\_reachability\_t  \* @flags READ\_ONLY  \*/  SAI\_SWITCH\_ATTR\_FABRIC\_PORT\_REACHABILITY\_LIST | |

## Switch Statistics

Global device statistics for switch and fabric device types (comments show which device type the counter applies to). Retrieve fabric stats using “get\_switch\_stats” SAI APIs.

|  |
| --- |
| /\*\*  \* @brief Switch counter IDs in sai\_get\_switch\_stats() call  \*  \* @flags Contains flags  \*/  typedef enum \_sai\_switch\_stat\_t  {  /\*\* ECC discards [fabric] \*/  SAI\_SWITCH\_STAT\_ECC\_DROP,  /\*\* Reachability discards [switch | fabric] \*/  SAI\_SWITCH\_STAT\_REACHABILITY\_DROP,  /\*\* Congestion related high watermark [switch] \*/  SAI\_SWITCH\_STAT\_HIGHEST\_QUEUE\_CONGESTION\_LEVEL,  /\*\* discards not counted in other switch stat type [switch | fabric] \*/  SAI\_SWITCH\_STAT\_GLOBAL\_DROP,  /\*\* Custom range base value \*/  SAI\_SWITCH\_STAT\_CUSTOM\_RANGE\_BASE = 0x10000000  } sai\_switch\_stat\_t; |

## Port Attributes, Types, and APIs

## New Port Types

|  |
| --- |
| /\*\*  \* @brief Port Type  \*  \* @type sai\_port\_type\_t  \* @flags CREATE\_ONLY  \*/  SAI\_PORT\_ATTR\_TYPE = SAI\_PORT\_ATTR\_START,  /\*\*  \* @brief Attribute data for #SAI\_PORT\_ATTR\_TYPE  \*/  typedef enum \_sai\_port\_type\_t  {  /\*\* Actual port. N.B. Different from the physical port. \*/  SAI\_PORT\_TYPE\_LOGICAL,  /\*\* CPU Port \*/  SAI\_PORT\_TYPE\_CPU,  /\*\* System Port \*/  SAI\_PORT\_TYPE\_SYSTEM,  /\*\* Fabric Port \*/  SAI\_PORT\_TYPE\_FABRIC  } sai\_port\_type\_t; |

## 

## Port Connectivity Attributes for System and Fabric ports

It is important to note that some port attributes are not valid for certain port types. For example, SAI\_PORT\_ATTR\_HW\_LANE\_LIST is not valid for system ports.

Port configuration can be retrieved for port types System, and Fabric by using the SAI\_PORT\_ATTR\_CONFIG\_INFO port attribute.

For Fabric ports, two new read only attributes are added to retrieve the attached switch id and port index for each fabric port. These can be used to check connectivity.

|  |
| --- |
| /\*\*  \* @brief Port Configuration Attribute Value  \*/  typedef union \_sai\_port\_config\_info\_t  {  /\*\* Configuration of system port \*/  sai\_sysport\_config\_t system\_port;  /\*\* Configuration of fabric port \*/  sai\_fabric\_port\_config\_t fabric\_port;  } sai\_port\_config\_info\_t;  /\*\*  \* @brief Port Configuration Information  \*  \* @type sai\_port\_config\_t  \* @flags READ\_ONLY  \*/  SAI\_PORT\_ATTR\_CONFIG\_INFO  /\*\*  \* @brief Attached switch id.  \*  \* Signifies the destination switch id  \*  \* @type sai\_uint32\_t  \* @flags READ\_ONLY  \*/  SAI\_PORT\_ATTR\_FABRIC\_ATTACHED\_SWITCH\_ID  /\*\*  \* @brief Attached Port Index.  \*  \* Signifies the destination port index  \*  \* @type sai\_uint32\_t  \* @flags READ\_ONLY  \*/  SAI\_PORT\_ATTR\_FABRIC\_ATTACHED\_PORT\_INDEX |

## Fabric Port Error Status Attributes

The following port error status attribute can give more info on errors occurred on port.

|  |  |
| --- | --- |
| /\*\*  \* @brief Port Down Error Status  \*  \* @type sai\_port\_err\_status\_t  \* @flags READ\_ONLY  \*/  SAI\_PORT\_ATTR\_ERR\_STATUS  /\*\*  \* @brief Attribute data for #SAI\_PORT\_ATTR\_ERR\_STATUS  \*/  typedef enum \_sai\_port\_err\_status\_t  {  /\*\* Data Unit CRC Error \*/  SAI\_PORT\_ERR\_DATA\_UNIT\_CRC\_ERROR,  /\*\* Data Unit Size Error \*/  SAI\_PORT\_ERR\_DATA\_UNIT\_SIZE,  /\*\* Data Unit Misalignment Error \*/  SAI\_PORT\_ERR\_DATA\_UNIT\_MISALIGNMENT\_ERROR,  /\*\* Uncorrectable RS-FEC code word error \*/  SAI\_PORT\_ERR\_CODE\_GROUP\_ERROR,  /\*\* SerDes Signal is out of sync \*/  SAI\_PORT\_ERR\_SIGNAL\_LOCAL\_ERROR,  /\*\* Port is not accepting reachability data units \*/  SAI\_PORT\_ERR\_NO\_RX\_REACHABILITY,  /\*\* Rate of data units with CRC errors passed its threshold \*/  SAI\_PORT\_ERR\_CRC\_RATE    /\*\* Error remote fault indication \*/  SAI\_FABRIC\_PORT\_ERR\_REMOTE\_FAULT\_STATUS  } sai\_port\_err\_status\_t; | |

## SAI\_OBJECT\_TYPE\_LAG

The SAI\_LAG\_ATTR\_PORT\_LIST also supports the list of SAI\_PORT\_TYPE\_SYSTEM objects.

## New Ports Stats

New statistics for fabric ports on switch and fabric devices. Note that FEC statistics can also apply to other physical port types.

|  |  |
| --- | --- |
| /\*\*  \* @brief Port counter IDs in sai\_get\_port\_stats() call  \*  \* @flags Contains flags  \*/  typedef enum \_sai\_port\_stat\_t  {  SAI\_PORT\_STAT\_IF\_IN\_OCTETS,  ...  SAI\_PORT\_STAT\_IF\_IN\_ERRORS,  ...  SAI\_PORT\_STAT\_IF\_OUT\_OCTETS,  ...  SAI\_PORT\_STAT\_IF\_IN\_FEC\_CORRECTABLE\_FRAMES,  SAI\_PORT\_STAT\_IF\_IN\_FEC\_UNCORRECTABLE\_FRAMES,  SAI\_PORT\_STAT\_IF\_IN\_FEC\_SYMBOL\_ERRORS,  ...  /\*\* SAI stats for fabric ports data units \*/  SAI\_PORT\_STAT\_IF\_IN\_FABRIC\_DATA\_UNITS,  SAI\_PORT\_STAT\_IF\_OUT\_FABRIC\_DATA\_UNITS,  ...  } sai\_port\_stat\_t; | |

## Queue Attributes, Types, and APIs

## New Queue Types

|  |
| --- |
| /\*\*  \* @brief Enum defining Queue types.  \*/  typedef enum \_sai\_queue\_type\_t  {  /\*\* H/w Queue for all types of traffic \*/  SAI\_QUEUE\_TYPE\_ALL = 0x00000000,  /\*\* H/w Egress Unicast Queue \*/  SAI\_QUEUE\_TYPE\_UNICAST = 0x00000001,  /\*\* H/w Multicast Egress (Broadcast, Unknown unicast, Multicast) Queue \*/  SAI\_QUEUE\_TYPE\_MULTICAST = 0x00000002,  /\*\* H/w Virtual Output Queue (VoQ). This queue is ingress unicast queue \*/  SAI\_QUEUE\_TYPE\_UNICAST\_VOQ = 0x00000003,  /\*\* H/w Fabric Queue. \*/  SAI\_QUEUE\_TYPE\_FABRIC\_TX = 0x00000004,  /\* Custom range base value \*/  SAI\_QUEUE\_TYPE\_CUSTOM\_RANGE\_BASE = 0x10000000  } sai\_queue\_type\_t; |

## VoQ Attributes

Newly defined VOQ type (SAI\_QUEUE\_TYPE\_UNICAST\_VOQ) is meant to use existing Queue attributes and flows. VOQs are created during System Port creation. It uses the speed attribute passed during system port creation to set defaults for queue management. From that point on, existing methods are used to set queue thresholds (e.g. WRED Profile).  
Note: association to Scheduler is configured on the queues of the local Port, same as a non VOQ system.

## Fabric Queue Attributes

Fabric queues (internal Tx FIFOs) are used for retrieving queue statistics for fabric port types (no distinct attributes).

## Modification to queue stats

|  |
| --- |
| /\*\*  \* @brief Enum defining statistics for Queue.  \*/  typedef enum \_sai\_queue\_stat\_t  {  /\*\* Get/set tx packets count [uint64\_t] \*/  /\*\* [relevance for VOQ system] \*/  SAI\_QUEUE\_STAT\_PACKETS = 0x00000000, [VOQ |QUEUE]  /\*\* Get/set tx bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_BYTES = 0x00000001, [VOQ |QUEUE]  /\*\* Get/set dropped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_DROPPED\_PACKETS = 0x00000002, [VOQ |QUEUE]  /\*\* Get/set dropped bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_DROPPED\_BYTES = 0x00000003, [VOQ |QUEUE]  /\*\* Get/set green color tx packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_GREEN\_PACKETS = 0x00000004, [VOQ]  /\*\* Get/set green color tx bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_GREEN\_BYTES = 0x00000005, [VOQ]  /\*\* Get/set green color dropped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_GREEN\_DROPPED\_PACKETS = 0x00000006, [VOQ]  /\*\* Get/set green color dropped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_GREEN\_DROPPED\_BYTES = 0x00000007, [VOQ]  /\*\* Get/set yellow color tx packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_YELLOW\_PACKETS = 0x00000008, [VOQ]  /\*\* Get/set yellow color tx bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_YELLOW\_BYTES = 0x00000009, [VOQ]  /\*\* Get/set yellow color drooped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_YELLOW\_DROPPED\_PACKETS = 0x0000000a, [VOQ]  /\*\* Get/set yellow color dropped bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_YELLOW\_DROPPED\_BYTES = 0x0000000b, [VOQ]  /\*\* Get/set red color tx packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_RED\_PACKETS = 0x0000000c, [VOQ]  /\*\* Get/set red color tx bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_RED\_BYTES = 0x0000000d, [VOQ]  /\*\* Get/set red color dropped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_RED\_DROPPED\_PACKETS = 0x0000000e, [VOQ]  /\*\* Get/set red color drooped bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_RED\_DROPPED\_BYTES = 0x0000000f, [VOQ]  /\*\* Get/set WRED green color dropped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_GREEN\_WRED\_DROPPED\_PACKETS = 0x00000010, [VOQ]  /\*\* Get/set WRED green color dropped bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_GREEN\_WRED\_DROPPED\_BYTES = 0x00000011, [VOQ]  /\*\* Get/set WRED yellow color dropped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_YELLOW\_WRED\_DROPPED\_PACKETS = 0x00000012, [VOQ]  /\*\* Get/set WRED yellow color dropped bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_YELLOW\_WRED\_DROPPED\_BYTES = 0x00000013, [VOQ]  /\*\* Get/set WRED red color dropped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_RED\_WRED\_DROPPED\_PACKETS = 0x00000014, [VOQ]  /\*\* Get/set WRED red color dropped bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_RED\_WRED\_DROPPED\_BYTES = 0x00000015, [VOQ]  /\*\* Get/set WRED dropped packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_WRED\_DROPPED\_PACKETS = 0x00000016, [VOQ]  /\*\* Get/set WRED red dropped bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_WRED\_DROPPED\_BYTES = 0x00000017, [VOQ]  /\*\* Get current queue occupancy in bytes [uint64\_t] \*/  SAI\_QUEUE\_STAT\_CURR\_OCCUPANCY\_BYTES = 0x00000018, [VOQ]  /\*\* Get watermark queue occupancy in bytes [uint64\_t] \*/  SAI\_QUEUE\_STAT\_WATERMARK\_BYTES = 0x00000019, [VOQ| Fabric Queue]  /\*\* Get current queue shared occupancy in bytes [uint64\_t] \*/  SAI\_QUEUE\_STAT\_SHARED\_CURR\_OCCUPANCY\_BYTES = 0x0000001a,[N/A]  /\*\* Get watermark queue shared occupancy in bytes [uint64\_t] \*/  SAI\_QUEUE\_STAT\_SHARED\_WATERMARK\_BYTES = 0x0000001b, [N/A]  /\*\* Get/set WRED green color marked packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_GREEN\_WRED\_ECN\_MARKED\_PACKETS = 0x0000001c, [N/A]  /\*\* Get/set WRED green color marked bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_GREEN\_WRED\_ECN\_MARKED\_BYTES = 0x0000001d, [N/A]  /\*\* Get/set WRED yellow color marked packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_YELLOW\_WRED\_ECN\_MARKED\_PACKETS = 0x0000001e, [N/A]  /\*\* Get/set WRED yellow color marked bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_YELLOW\_WRED\_ECN\_MARKED\_BYTES = 0x0000001f, [N/A]  /\*\* Get/set WRED red color marked packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_RED\_WRED\_ECN\_MARKED\_PACKETS = 0x00000020, [N/A]  /\*\* Get/set WRED red color marked bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_RED\_WRED\_ECN\_MARKED\_BYTES = 0x00000021, [N/A]  /\*\* Get/set WRED marked packets count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_WRED\_ECN\_MARKED\_PACKETS = 0x00000022, [QUEUE]  /\*\* Get/set WRED red marked bytes count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_WRED\_ECN\_MARKED\_BYTES = 0x00000023, [QUEUE]  /\*\* Get/set incoming fabric data unit count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_IN\_FABRIC\_DATA\_UNITS = 0x00000024,  /\*\* Get outgoing fabric data unit count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_OUT\_FABRIC\_DATA\_UNITS = 0x00000025,  /\*\* Get dropped fabric data unit count [uint64\_t] \*/  SAI\_QUEUE\_STAT\_OUT\_DROPPED\_FABRIC\_DATA\_UNITS = 0x00000026,    /\*\* Custom range base value \*/  SAI\_QUEUE\_STAT\_CUSTOM\_RANGE\_BASE = 0x10000000  } sai\_queue\_stat\_t; |

## 2.5 New Neighbor Entry Attribute

In a VOQ system, data plane functionality is partitioned between Ingress Switch (where the packet enters, and Forwarding process is performed), and Egress Switch (where the packet is modified and transmitted). Egress device objects need to be synchronized with Ingress devices to provide coherent end-to-end data plane. Objects created at an Egress device will be referenced from all Ingress Switches.  
When a neighbor is discovered (ARP/NDP) on a remote Switch, the neighbor entry at the remote Switch must be created and referenced to the local Switch SAI.  
The remote Switch neighbor is realized as an OutLIF in its ASIC-DB, the remote Switch OutLIF index is provided using a new attribute (SAI\_NEIGHBOR\_ENTRY\_ATTR\_REMOTE\_OUTLIF) for the neighbor entry. The newly created neighbor will use this attribute along with the RIF, to route traffic properly to the remote Switch and determine its processing at the Egress Switch.

|  |  |
| --- | --- |
| /\*\*  \* @brief Attribute Id for SAI neighbor object  \*/  typedef enum \_sai\_neighbor\_entry\_attr\_t  {  ...  /\*\*  \* @brief Remote Out Logical Interface  \*  \* Defines the remote neighbor OUTLIF when the destination is on a remote line card  \*  \* @type sai\_uint32\_t  \* @flags CREATE\_AND\_SET  \* @default 0  \*/  SAI\_NEIGHBOR\_ENTRY\_ATTR\_REMOTE\_OUTLIF,  ...  } sai\_neighbor\_entry\_attr\_t; | |

# SAI Usage Examples

This section is intended to show the usage of newly defined types, attributes, and APIs. it is not intended to show flows of existing attributes and APIs.

## SAI Switch Create API

This is an existing API. Here is an example of how Ports and System-Ports config is passed to this API. This will allow all Port and System-Port information to be passed at one time during switch\_create (before SDK init). This is a simpler implementation.

|  |
| --- |
| sai\_sysport\_config\_t sysport\_config\_list[] = {  {  .sysport\_id = minigraph.sysport[0].sysport\_id,  .switch\_id = minigraph.sysport[0].switch\_id,  .core\_index = minigraph.sysport[0].core\_index,  .core\_port\_index = minigraph.sysport[0].core\_port\_index,  .speed = minigraph.sysport[0].speed,  .num\_voq = minigraph.sysport[0].num\_voq,  .base\_voq = minigraph.sysport[0].base\_voq  },  …  {  .sysport\_id = minigraph.sysport[31].sysport\_id,  .switch\_id = minigraph.sysport[31].switch\_id,  .core\_index = minigraph.sysport[31].core\_index,  .core\_port\_index = minigraph.sysport[31].core\_port\_index,  .speed = minigraph.sysport[31].speed,  .num\_voq = minigraph.sysport[31].num\_voq,  .base\_voq = minigraph.sysport[31].base\_voq  }  }  sai\_attribute\_t attr[] = {  ...  { SAI\_SWITCH\_ATTR\_SWITCH\_ID, .value.u32 = minigraph.switch\_id },  { SAI\_SWITCH\_ATTR\_MAX\_SYSTEM\_CORES, .value.u32 = minigraph.max\_cores },  { SAI\_SWITCH\_ATTR\_SYSTEM\_PORT\_CONFIG\_LIST, .value.sai\_sysport\_config\_list\_t =  { .count=minigraph.num\_sysports, .list = sysport\_config\_list } },  ...  }  rv = sai\_switch\_apis->create\_switch(&switch\_id, COUNTOF(attr), attr); |

## SAI Switch Get Attribute API

With front panel ports and fabric ports, mapping between SAI port objects and port indexes that the application is aware of can be done by getting the port objects’ associated HW\_LANE\_LIST. But with system ports the HW\_LANE\_LIST is not available so another method must be used. Below is an example of how the mapping could be done for system ports. The example also contains an alternate way for logical and fabric port mappings to be determined.

|  |
| --- |
| sai\_attribute\_t attr;  attr.id = SAI\_SWITCH\_ATTR\_NUMBER\_OF\_SYSTEM\_PORTS;  rv = sai\_switch\_api->get\_switch\_attribute(gSwitchId, 1, &attr);  m\_portCount = attr.value.u32;  attr.id = SAI\_SWITCH\_ATTR\_SYSTEM\_PORT\_LIST;  attr.value.objlist.count = (uint32\_t)sys\_port\_list.size();  attr.value.objlist.list = sys\_port\_list.data();  /\* get list of system ports \*/  rv = sai\_switch\_apis->get\_switch\_attribute(switchid, COUNTOF(attr), &attr);  port\_list = attr.value.objlist.list;  /\* How to map system port object to system port \*/  for (i=0; i<m\_portCount; i++) {  attr.id = SAI\_PORT\_ATTR\_CONFIG\_INFO;  rv = = sai\_port\_api->get\_port\_attribute(port\_list[i], 1, &attr);  /\* Use returned System Port ID for mapping \*/  obj\_map[i].port\_obj = port\_list[i];  obj\_map[i].port\_id = attr.value.port\_config.config.system\_port.sysport\_id;  }  attr.id = SAI\_SWITCH\_ATTR\_NUMBER\_OF\_ACTIVE\_PORTS;  rv = sai\_switch\_api->get\_switch\_attribute(gSwitchId, 1, &attr);  m\_portCount = attr.value.u32;  attr.id = SAI\_SWITCH\_ATTR\_PORT\_LIST;  attr.value.objlist.count = (uint32\_t)port\_list.size();  attr.value.objlist.list = port\_list.data();  /\* get list of logical ports \*/  rv = sai\_switch\_apis->get\_switch\_attribute(switchid, COUNTOF(attr), &attr);  port\_list = attr.value.objlist.list;  /\* How to map logical port object to logical port \*/  for (i=0; i<m\_portCount; i++) {  attr.id = SAI\_PORT\_ATTR\_CONFIG\_INFO;  rv = = sai\_port\_api->get\_port\_attribute(port\_list[i], 1, &attr);  /\* Use returned System Port ID for mapping \*/  obj\_map[i].port\_obj = port\_list[i];  obj\_map[i].port\_id = attr.value.port\_config.config.logical\_port.port\_id;  }  attr.id = SAI\_SWITCH\_ATTR\_NUMBER\_OF\_FABRIC\_PORTS;  rv = sai\_switch\_api->get\_switch\_attribute(gSwitchId, 1, &attr);  m\_portCount = attr.value.u32;  attr.id = SAI\_SWITCH\_ATTR\_FABRIC\_PORT\_LIST;  attr.value.objlist.count = (uint32\_t)fab\_port\_list.size();  attr.value.objlist.list = fab\_port\_list.data();  /\* get list of fabric ports \*/  rv = sai\_switch\_apis->get\_switch\_attribute(switchid, COUNTOF(attr), &attr);  port\_list = attr.value.objlist.list;  /\* How to map fabric port object to fabric port \*/  for (i=0; i<m\_portCount; i++) {  attr.id = SAI\_PORT\_ATTR\_CONFIG\_INFO;  rv = = sai\_port\_api->get\_port\_attribute(port\_list[i], 1, &attr);  /\* Use returned System Port ID for mapping \*/  obj\_map[i].port\_obj = port\_list[i];  obj\_map[i].port\_id = attr.value.port\_config.config.fabric\_port.port\_id;  } |

## SAI Port Create API

This is an existing port API that can be used to create local ports and system ports. Here is an example of how to create a single system port.

|  |
| --- |
| sai\_port\_config\_t port\_config = {  .system\_port.sysport\_id = mgraph.sysport[0].sysport\_id,  .system\_port.attached\_switch\_id = mgraph.sysport[0].switch\_id,  .system\_port.attached\_core\_index = mgraph.sysport[0].core\_index,  .system\_port.attached\_core\_port\_index = mgraph.sysport[0].core\_port\_index,  .system\_port.speed = mgraph.sysport[0].speed,  .system\_port.num\_voq = mgraph.sysport[0].num\_voq,  .system\_port.base\_voq\_index = mgraph.sysport[0].base\_voq  };    sai\_attribute\_t port\_attr[] = {  …  { SAI\_PORT\_ATTR\_TYPE, .value.s32 = SAI\_PORT\_TYPE\_SYSTEM },  { SAI\_PORT\_ATTR\_CONFIG\_INFO, {.value.port\_config = port\_config},  ...  };  rv = sai\_port\_apis->create\_port(&port\_obj,  switchid,  COUNTOF(port\_attr),  port\_attr); |

## SAI Port Bulk Object Create API

This is a new port API. Here is an example of how to create multiple system ports with one API call.

|  |
| --- |
| /\*\*  \* @brief Port methods table retrieved with sai\_api\_query()  \*/  typedef struct \_sai\_port\_api\_t  {  sai\_create\_port\_fn create\_port;  sai\_remove\_port\_fn remove\_port;  sai\_set\_port\_attribute\_fn set\_port\_attribute;  sai\_get\_port\_attribute\_fn get\_port\_attribute;  ...  sai\_bulk\_object\_create\_fn create\_ports;  sai\_bulk\_object\_remove\_fn remove\_ports;  ...  } sai\_port\_api\_t;  sai\_port\_config\_t port\_config[] = {  {.system\_port.sysport\_id = mgraph.sysport[0].sysport\_id,  .system\_port.attached\_switch\_id = mgraph.sysport[0].switch\_id,  .system\_port.attached\_core\_index = mgraph.sysport[0].core\_index,  .system\_port.attached\_core\_port\_index = mgraph.sysport[0].core\_port\_index,  .system\_port.speed = mgraph.sysport[0].speed,  .system\_port.num\_voq = mgraph.sysport[0].num\_voq,  .system\_port.base\_voq\_index = mgraph.sysport[0].base\_voq},  ...  {.system\_port.sysport\_id = mgraph.sysport[9].sysport\_id,  .system\_port.attached\_switch\_id = mgraph.sysport[9].switch\_id,  .system\_port.attached\_core\_index = mgraph.sysport[9].core\_index,  .system\_port.attached\_core\_port\_index = mgraph.sysport[9].core\_port\_index,  .system\_port.speed = mgraph.sysport[9].speed,  .system\_port.num\_voq = mgraph.sysport[9].num\_voq,  .system\_port.base\_voq\_index = mgraph.sysport[9].base\_voq}  };  sai\_attribute\_t sysport\_list\_attr[] = {  …  { SAI\_PORT\_ATTR\_TYPE, .value.s32 = SAI\_PORT\_TYPE\_SYSTEM },  { SAI\_PORT\_ATTR\_CONFIG\_INFO, .value.port\_config = port\_config[0] },  ...  { SAI\_PORT\_ATTR\_CONFIG\_INFO, .value.port\_config = port\_config[9] },  ...  };  rv = sai\_port\_apis->create\_ports(switchid,  minigraph.sysport\_count,  &num\_sysport\_attr,  sysport\_list\_attr,  SAI\_BULK\_OP\_ERROR\_MODE\_IGNORE\_ERROR,  sysport\_list\_object\_id,  sysport\_list\_status); |

## 

# Appendix

## SAI\_SWITCH\_ATTR\_PORT\_CONFIG\_LIST

This switch attribute structure defines a list of port parameters passed to the create\_switch API. Used for front panel for lane mapping. Defined in “saitypes.h”.

This switch attribute is out of scope from the VoQ switch proposal but is added here for completeness and consistency with the configuration of newly introduced port types (system and fabric ports).

|  |
| --- |
| /\*\*  \* @brief Maximum Number of Port Lanes  \*/  #define SAI\_MAX\_PORT\_LANES 8  typedef struct \_sai\_fp\_port\_config\_t {  int port\_id; // logical port id  int speed; // port speed  int count; // actual lanes used  uint32\_t lanes[SAI\_MAX\_PORT\_LANES]; // fixed max number of lanes that can be utilized  } sai\_fp\_port\_config\_t;  /\*\*  \* @brief Port Configuration Attribute Value  \*/  typedef union \_sai\_port\_config\_t  {  /\*\* Configuration of front panel port \*/  sai\_fp\_port\_config\_t logical\_port;  ...  } sai\_port\_config\_t;  typedef struct \_sai\_fp\_port\_config\_list\_t  {  /\*\* Number of entries in the list \*/  uint32\_t count;  /\*\* Port list \*/  sai\_fp\_port\_config\_t \*list;  } sai\_fp\_port\_config\_list\_t;  /\*\*  \* @brief Port config list.  \*  \* @type sai\_fp\_port\_config\_list\_t  \* @flags CREATE\_ONLY  \* @default empty  \*/  SAI\_SWITCH\_ATTR\_PORT\_CONFIG\_LIST |

Example usage:

|  |
| --- |
| sai\_fp\_port\_config\_t port\_config\_list[] = {  {  .port\_index = port.ini[0].index,  .speed = port.ini[0].speed,  .lane\_list.count = port.ini[0].lane\_count,  .lane\_list.list = &port.ini[0].lane\_list  },  …  {  .port\_index = port.ini[24].index,  .speed = port.ini[24].speed,  .lane\_list.count = port.ini[24].lane\_count,  .lane\_list.list = &port.ini[24].lane\_list  }  }  sai\_attribute\_t attr[] = {  ...  { SAI\_SWITCH\_ATTR\_PORT\_CONFIG\_LIST, .value.sai\_port\_config\_list =  { .count=minigraph.num\_ports, .list = port\_config\_list } },  ...  }  rv = sai\_switch\_apis->create\_switch(&switch\_id, COUNTOF(attr), attr); |